

RECAP

B649


Parallel Architectures and Programming


Parallel Programming

- A bag of tricks
 - ★ less structured and categorized than “sequential” programming
 - ★ less well understood in theoretical terms
 - ★ still evolving
- Trends
 - ★ at an inflection point?
 - ★ legacy code and new code
 - ★ productivity a major concern

PRODUCTIVITY IN HPC

High Productivity Computing Systems

INFORMATION PROCESSING TECHNIQUES OFFICE		THRUST AREAS	PROGRAMS	SOLICITATIONS	PERSONNEL
IPTO >> Programs >> High Productivity Computing Systems (HPCS)			QUICK LINKS:		
			COGNITIVE SYSTEMS ▾		
			Open Solicitations BAA 09-03 Machine Reading BAA 09-40 Deep Learning		
			COMMAND & CONTROL ▾		
			Open Solicitations BAA 08-36 ULTRA-Vis		
			HIGH PRODUCTIVITY COMPUTING ▾		
			No Open Solicitations		
			LANGUAGE PROCESSING		
			No Open Solicitations		
			SENSORS & PROCESSING		
			Open Solicitations BAA 09-15 Retriever		
			EMERGING TECHNOLOGIES		
			Open Solicitations BAA 08-34 IPTO Office-Wide		
			IPTO IS LOOKING FOR:		
			Program Managers and New Ideas		

	<h2>High Productivity Computing Systems (HPCS)</h2> <p>Program Manager: Dr. Charles Holland</p> <p>Mission:</p> <p>Conduct a focused research and development program that creates a new generation of high productivity computing systems. These computing systems will comprise software tools, architectures, and hardware components. These systems will improve by orders of magnitude the effectiveness of humans and computers in solving the problems in the high performance computing domain.</p> <p>Create economically viable high productivity computing systems for the national security and industrial user communities. These systems must have the following attributes:</p> <ul style="list-style-type: none">• Performance: Improve the computational efficiency and reduce the execution time of critical national security applications.• Programmability: Reduce cost and time of developing HPCS applications.• Portability: Insulate HPCS application software from system specifics.• Robustness: Improve reliability and reduce vulnerability to intentional attacks.
--	--

<p>Programs Home</p> <p>Overview</p> <ul style="list-style-type: none">▶ Mission▶ Background▶ Vision <p>Technical Program</p> <ul style="list-style-type: none">▶ Assessment▶ Challenges▶ Goals▶ Objectives▶ Program Plan
--

High Productivity Computing Systems



High Productivity Computer Systems

Providing a New Generation of Economically Viable High Productivity Computing Systems

The DARPA High Productivity Computing Systems is focused on providing a new generation of economically viable high productivity computing systems for national security and for the industrial user community. HPCS program researchers have initiated a fundamental reassessment of how we define and measure performance, programmability, portability, robustness and ultimately, productivity in the HPC domain.

Introduction

Meetings

References

Working Groups

- Management
- Execution Time Modeling
- Development Time Exp
- Existing Codes Analysis
- Workflows, Models & Metrics
- Benchmarks
- Prog Models & Definitions
- Test & Spec Environment

Participants

Benchmarks

- HPC Challenge
- HPC Challenge Award
- Compact Apps
- Kernel Matrix
- NAS Benchmarks

Home

[Monthly Meeting Info](#)

ACCESS GRID

A Project Sponsored by



Productivity and Performance

DEFINING AND MEASURING THE PRODUCTIVITY OF PROGRAMMING LANGUAGES

Ken Kennedy¹
Charles Koelbel¹
Robert Schreiber²

Abstract

The goal of programming support systems is to make it possible for application developers to produce software faster, without any degradation in software quality. However, it is essential that this goal must not be achieved at the cost of performance: programs written in a high-level language and intended to solve large problems on highly parallel machines must not be egregiously less efficient than the same applications written in a lower-level language. Because this has been a traditional stumbling block for high-level languages, metrics for productivity analysis must explore the trade-off between programming effort and performance.

To that end, we propose the use of two dimensionless ratios: relative power and relative efficiency to measure

1 Introduction

The overall objective of programming support systems is to make it possible to produce software faster with the same workforce, with no degradation, and possibly an improvement, in software quality. Generally, there are two ways to approach this goal. First, we can increase the effectiveness of individual application developers by providing programming languages and tools that enhance programming productivity. Secondly, we can broaden the community of application developers by making programming more accessible. As it happens, the use of higher-level languages and programming interfaces supports both these strategies: by incorporating a higher level of abstraction, such languages make application development both easier and faster. (For the purposes of this paper, we will define “programming language” to encompass the entire toolset – language, compiler, debugger, tuning tools – associated with the language.)

We must, however, ensure that these advantages do not come at the cost of performance. Programs written in a high-level language and intended to solve large problems on highly parallel machines must not be egregiously less efficient than the same applications written in a lower-level language. If they are, then the language is unlikely to be accepted. Because this has been a traditional stumbling block for high-level languages, our productivity analysis must incorporate metrics of both programming effort and performance. Furthermore, these metrics must

Ken Kennedy, Charles Koelbel and Robert Schreiber. Defining and Measuring the Productivity of Programming Languages, International Journal of High Performance Computing Applications, Vol. 18, No. 4, 441-448 (2004).

Productivity and Performance

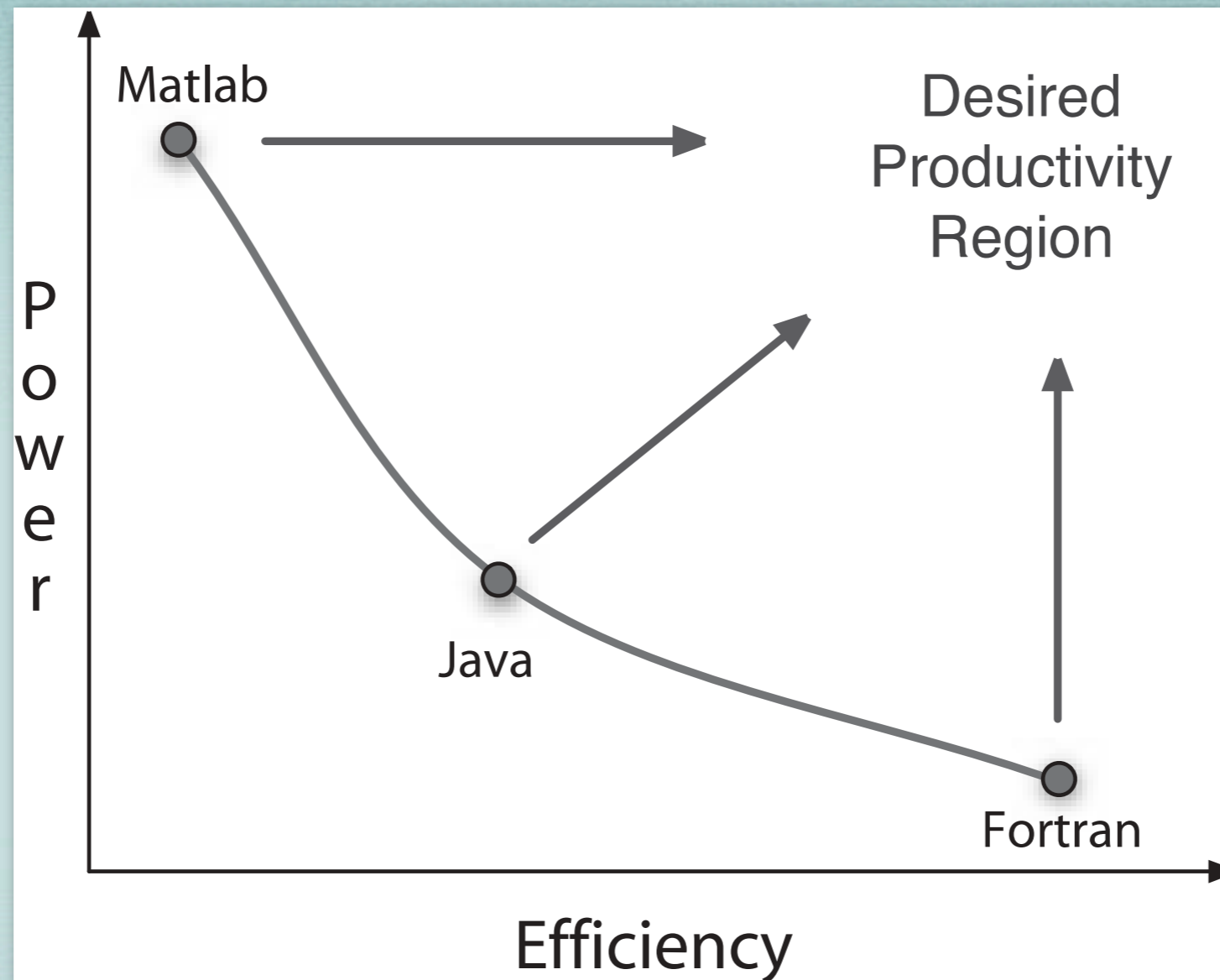


Fig. 1 Power–efficiency graph.

Ken Kennedy, Charles Koelbel and Robert Schreiber. Defining and Measuring the Productivity of Programming Languages, International Journal of High Performance Computing Applications, Vol. 18, No. 4, 441-448 (2004).

Case Studies in HPC

	ASC Codes	MP Codes
# of projects	5	5
Environment	Academia (ASC-Alliance projects)	Mission Partners (DoD, DOE, NASA)
Classified	No	Some
Code size	200-600 KLOC	80-760 KLOC
Type	Coupled multi-physics applications	Single physics to coupled multi-physics and engineering

Table 1. Types of projects examined.

Jeffrey C. Carver, Lorin M. Hochstein, Richard P. Kendall, Taiga Nakamura, Marvin V. Zelkowitz, Victor R. Basili, and Douglass E. Post. **Observations about Software Development for High End Computing**, CTWatch Quarterly, 2(4A), November 2006.

Case Studies in HPC

	ASC Codes	MP Codes
Type	Ongoing	Retrospective
Interviewees	Technical leads	Projects leads, project staff
Overview	<ol style="list-style-type: none"> 1. Pre-interview questionnaire 2. Telephone interview 3. Generate summary document 4. Send summary document for approval/comments 5. Generate synthesis report across all projects 6. Send synthesis report to all centers for approval/ comments 	<ol style="list-style-type: none"> 1. Identify project and sponsors 2. Negotiate case study participation 3. Pre-interview questionnaire 4. On-site interview 5. Initial list of findings 6. Follow-up 7. Write report
Focus	<ul style="list-style-type: none"> • Product: attributes, machine target, history • Project organization: structure, staff, configuration management • Development activities: adding new features, testing, tuning, debugging, porting, effort distribution, bottlenecks, achieving performance • Programming models and productivity: choice of model, adoption of language, productivity measures 	<ul style="list-style-type: none"> • Goals, requirements, deliverables • Project characteristics, structure, organization and risks • Code Characteristics • Staffing • Workflow Management • V&V, Testing • Success Measures • Lessons Learned

Table 3. Methodology.

Jeffrey C. Carver, Lorin M. Hochstein, Richard P. Kendall, Taiga Nakamura, Marvin V. Zelkowitz, Victor R. Basili, and Douglass E. Post. **Observations about Software Development for High End Computing**, CTWatch Quarterly, 2(4A), November 2006.

HPC Case Studies: Cases

	FALCON	HAWK	CONDOR	EAGLE	NENE
Application Domain	Product Performance	Manufacturing	Product Performance	Signal Processing	Process Modeling
Duration	~ 10 years	~ 6 years	~ 20 years	~ 3 years	~ 25 years
# of Releases	9 (production)	1	7	1	?
Staffing	15 FTEs	3 FTEs	3-5 FTEs	3 FTEs	~10 FTEs (100's of contributors)
Customers	< 50	10s	100s	None	~ 100,000
Code Size	~ 405,000 LOC	~ 134,000 LOC	~200,000 LOC	< 100,000 LOC	750,000 LOC
Primary Languages	F77 (24%), C (12%)	C++ (67%), C (18%)	F77 (85%)	C++, Matlab	F77 (95%)
Other Languages	F90, Python, Perl, ksh/csh/sh	Python, F90	F90, C, Slang	Java Libraries	C
Target Hardware	Parallel Supercomputer	Parallel Supercomputer	PCs to Parallel Supercomputer	Embedded Hardware	PCs to Parallel Supercomputer
Status	Production	Production Ready	Production	Demonstration Code	Production

Jeffrey C. Carver, Richard P. Kendall, Susan E. Squires, and Douglass E. Post. **Software Development Environment for Scientific and Engineering Software: A Series of Case Studies**, In *Proceedings of the 29th International Conference on Software Engineering (ICSE)*, pages 550–559, 2007.

HPC Case Studies: Lessons

Jeffrey C. Carver, Richard P. Kendall, Susan E. Squires, and Douglass E. Post. **Software Development Environment for Scientific and Engineering Software: A Series of Case Studies**, In *Proceedings of the 29th International Conference on Software Engineering (ICSE)*, pages 550–559, 2007.

HPC Case Studies: Lessons

- Verification and validation is extremely difficult

Jeffrey C. Carver, Richard P. Kendall, Susan E. Squires, and Douglass E. Post. **Software Development Environment for Scientific and Engineering Software: A Series of Case Studies**, In *Proceedings of the 29th International Conference on Software Engineering (ICSE)*, pages 550–559, 2007.

HPC Case Studies: Lessons

- Verification and validation is extremely difficult
- Primary language does not change over time

Jeffrey C. Carver, Richard P. Kendall, Susan E. Squires, and Douglass E. Post. **Software Development Environment for Scientific and Engineering Software: A Series of Case Studies**, In *Proceedings of the 29th International Conference on Software Engineering (ICSE)*, pages 550–559, 2007.

HPC Case Studies: Lessons

- Verification and validation is extremely difficult
- Primary language does not change over time
- The use of higher-level languages is low

Jeffrey C. Carver, Richard P. Kendall, Susan E. Squires, and Douglass E. Post. **Software Development Environment for Scientific and Engineering Software: A Series of Case Studies**, In *Proceedings of the 29th International Conference on Software Engineering (ICSE)*, pages 550–559, 2007.

HPC Case Studies: Lessons

- Verification and validation is extremely difficult
- Primary language does not change over time
- The use of higher-level languages is low
- Developers prefer UNIX command-line over IDE

Jeffrey C. Carver, Richard P. Kendall, Susan E. Squires, and Douglass E. Post. **Software Development Environment for Scientific and Engineering Software: A Series of Case Studies**, In *Proceedings of the 29th International Conference on Software Engineering (ICSE)*, pages 550–559, 2007.

HPC Case Studies: Lessons

- Verification and validation is extremely difficult
- Primary language does not change over time
- The use of higher-level languages is low
- Developers prefer UNIX command-line over IDE
- Externally developed software is a risk

Jeffrey C. Carver, Richard P. Kendall, Susan E. Squires, and Douglass E. Post. **Software Development Environment for Scientific and Engineering Software: A Series of Case Studies**, In *Proceedings of the 29th International Conference on Software Engineering (ICSE)*, pages 550–559, 2007.

HPC Case Studies: Lessons

- Verification and validation is extremely difficult
- Primary language does not change over time
- The use of higher-level languages is low
- Developers prefer UNIX command-line over IDE
- Externally developed software is a risk
- Performance competes with other important goals
 - ★ correctness, performance, portability, maintainability

Jeffrey C. Carver, Richard P. Kendall, Susan E. Squires, and Douglass E. Post. **Software Development Environment for Scientific and Engineering Software: A Series of Case Studies**, In *Proceedings of the 29th International Conference on Software Engineering (ICSE)*, pages 550–559, 2007.

HPC Case Studies: Lessons

- Verification and validation is extremely difficult
- Primary language does not change over time
- The use of higher-level languages is low
- Developers prefer UNIX command-line over IDE
- Externally developed software is a risk
- Performance competes with other important goals
 - ★ correctness, performance, portability, maintainability
- Agile methodologies are better accepted

Jeffrey C. Carver, Richard P. Kendall, Susan E. Squires, and Douglass E. Post. **Software Development Environment for Scientific and Engineering Software: A Series of Case Studies**, In *Proceedings of the 29th International Conference on Software Engineering (ICSE)*, pages 550–559, 2007.

HPC Case Studies: Lessons

- Verification and validation is extremely difficult
- Primary language does not change over time
- The use of higher-level languages is low
- Developers prefer UNIX command-line over IDE
- Externally developed software is a risk
- Performance competes with other important goals
 - ★ correctness, performance, portability, maintainability
- Agile methodologies are better accepted
- Multidisciplinary teams are important to success

Jeffrey C. Carver, Richard P. Kendall, Susan E. Squires, and Douglass E. Post. **Software Development Environment for Scientific and Engineering Software: A Series of Case Studies**, In *Proceedings of the 29th International Conference on Software Engineering (ICSE)*, pages 550–559, 2007.

HPC Case Studies: Lessons

- Verification and validation is extremely difficult
- Primary language does not change over time
- The use of higher-level languages is low
- Developers prefer UNIX command-line over IDE
- Externally developed software is a risk
- Performance competes with other important goals
 - ★ correctness, performance, portability, maintainability
- Agile methodologies are better accepted
- Multidisciplinary teams are important to success
- Success or failure depends on keeping customers satisfied

Jeffrey C. Carver, Richard P. Kendall, Susan E. Squires, and Douglass E. Post. **Software Development Environment for Scientific and Engineering Software: A Series of Case Studies**, In *Proceedings of the 29th International Conference on Software Engineering (ICSE)*, pages 550–559, 2007.

Measuring Productivity in HPC

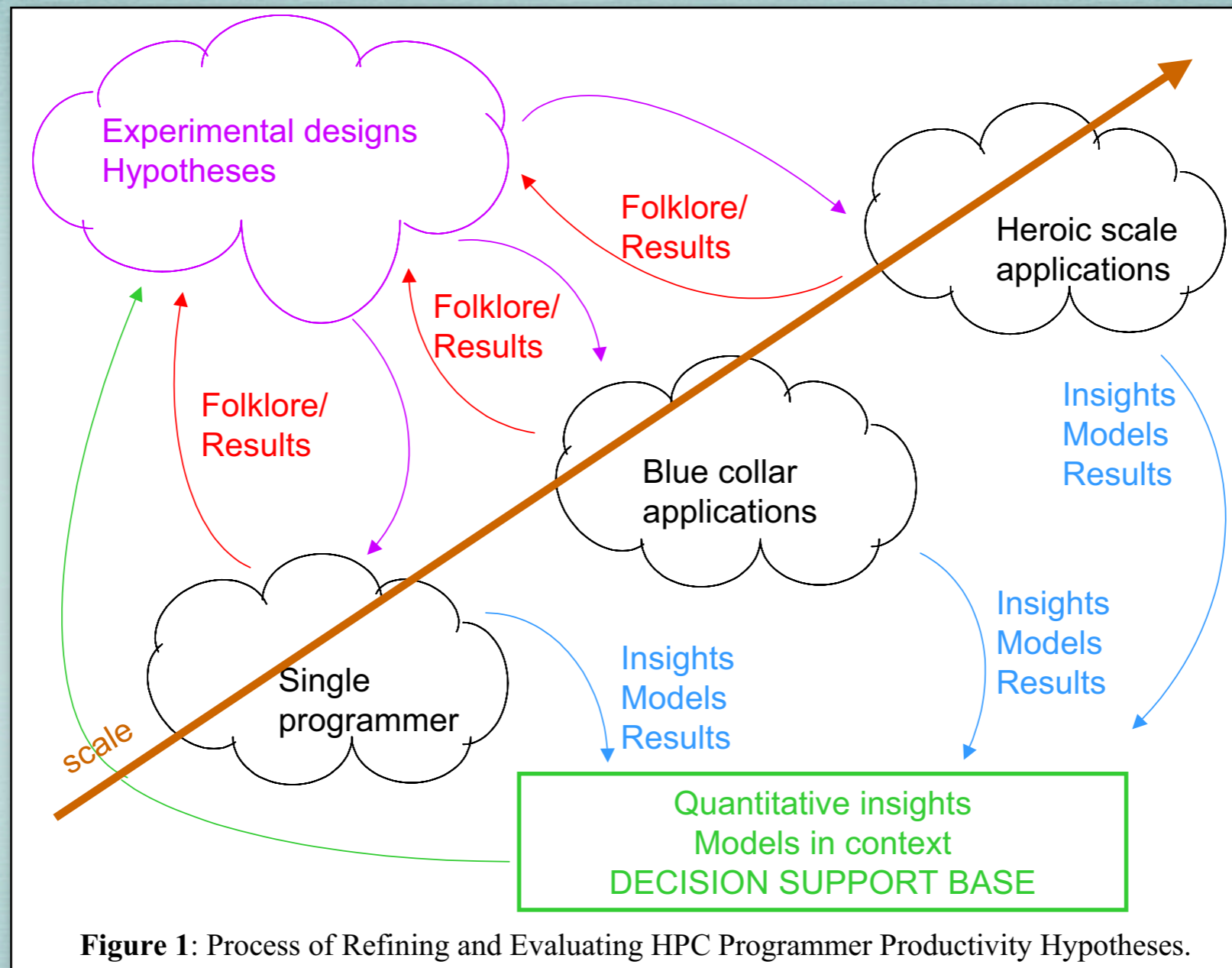


Figure 1: Process of Refining and Evaluating HPC Programmer Productivity Hypotheses.

Lorin Hochstein, Jeff Carver, Forrest Shull, Sima Asgari, and Victor Basili. **Parallel Programmer Productivity: A Case Study of Novice Parallel Programmers**, In *Proceedings of the 2005 ACM/IEEE Conference on High Performance Networking and Computing (SC '05)*, 2005.

Measuring Productivity in HPC

	Serial	MPI	OpenMP	Co-Array Fortran	StarP	XMT
Nearest-Neighbor Type Problems						
Game of Life	C3A3	C3A3 C0A1 C1A1	C3A3			
Grid of Resistors	C2A2	C2A2	C2A2		C2A2	
Sharks & Fishes		C6A2	C6A2	C6A2		
Laplace's Eq.		C2A3			P2A3	
SWIM			C0A2			
Broadcast Type Problems						
LU Decomposition			C4A1			
Parallel Mat-vec					C3A4	
Quantum Dynamics		C7A1				
Embarrassingly Parallel Type Problems						
Buffon-Laplace Needle		C2A1 C3A1	C2A1 C3A1		C2A1 C3A1	
<i>(Miscellaneous Problem Types)</i>						
Parallel Sorting		C3A2	C3A2		C3A2	
Array Compaction						C5A1
Randomized Selection						C5A2

Table 1: Matrix describing the problem space of HPC studies being run. Columns show the parallel programming model used. Rows show the assignment, grouped by communication pattern required. Each study is indicated with a label CxAy, identifying the participating class (C) and the assignment (A). Studies analyzed in this paper are grey-shaded.

Lorin Hochstein, Jeff Carver, Forrest Shull, Sima Asgari, and Victor Basili. **Parallel Programmer Productivity: A Case Study of Novice Parallel Programmers**, In *Proceedings of the 2005 ACM/IEEE Conference on High Performance Networking and Computing (SC '05)*, 2005.

Measuring Productivity in HPC

Programming Model	Effort (person-hrs)
Serial	mean 4.4, sd 4.3, n=15
MPI	mean 10.7, sd 8.9, n=16
OpenMP	mean 5.0, sd 3.5, n=16

Table 4: The Mean and standard deviation of the total effort along with the number of subjects is shown for each programming model. All data sets are for C implementations of the Game of Life for data set C3A3.

Lorin Hochstein, Jeff Carver, Forrest Shull, Sima Asgari, and Victor Basili. **Parallel Programmer Productivity: A Case Study of Novice Parallel Programmers**, In *Proceedings of the 2005 ACM/IEEE Conference on High Performance Networking and Computing (SC '05)*, 2005.

Productivity with MPI

Abstract— There is widespread belief in the computer science community that MPI is a difficult and time-intensive approach to developing parallel software. Nevertheless, MPI remains the dominant programming model for HPC systems, and many projects have made effective use of it. It remains unknown how much impact the use of MPI truly has on the productivity of computational scientists.

In this paper, we examine a mature, ongoing HPC project, the Flash Center at the University of Chicago, to understand how MPI is used and to estimate the time that programmers spend on MPI-related issues during development. Our analysis is based on an examination of the source code, version control history, and regression testing history of the software. Based on our study, we estimate that about 20% of the development effort is related to MPI. This implies a maximum productivity improvement of 25% for switching to an alternate parallel programming model.

Keywords: MPI, debugging, effort, productivity, case study

Lorin Hochstein, Forrest Shull, Lynn B. Reid. **The Role of MPI in Development Time: A Case Study**, In *Proceedings of the 2008 ACM/IEEE Conference on High Performance Networking and Computing (SC '08)*, 2008.

Productivity with MPI

TABLE I. SIZE OF CODE BASE

Language	With PARAMESH		Without PARAMESH	
	SLOC	% of total	SLOC	% of total
FORTRAN	377,149	87.2%	148,118	80.3%
C	29,058	6.7%	11,661	6.3%
Parameter (FLASH)	16,566	3.8%	16,566	9.0%
Config (FLASH)	3,841	0.9%	3,841	2.1%
Make	2,247	0.5%	1,403	0.8%
Perl	1,753	0.4%	1,753	1.0%
Python	1,576	0.4%	889	0.5%
Shell	475	0.1%	221	0.1%
<i>Total</i>	<i>432,665</i>	<i>100%</i>	<i>184,452</i>	<i>100%</i>

Lorin Hochstein, Forrest Shull, Lynn B. Reid. **The Role of MPI in Development Time: A Case Study**, In *Proceedings of the 2008 ACM/IEEE Conference on High Performance Networking and Computing (SC '08)*, 2008.

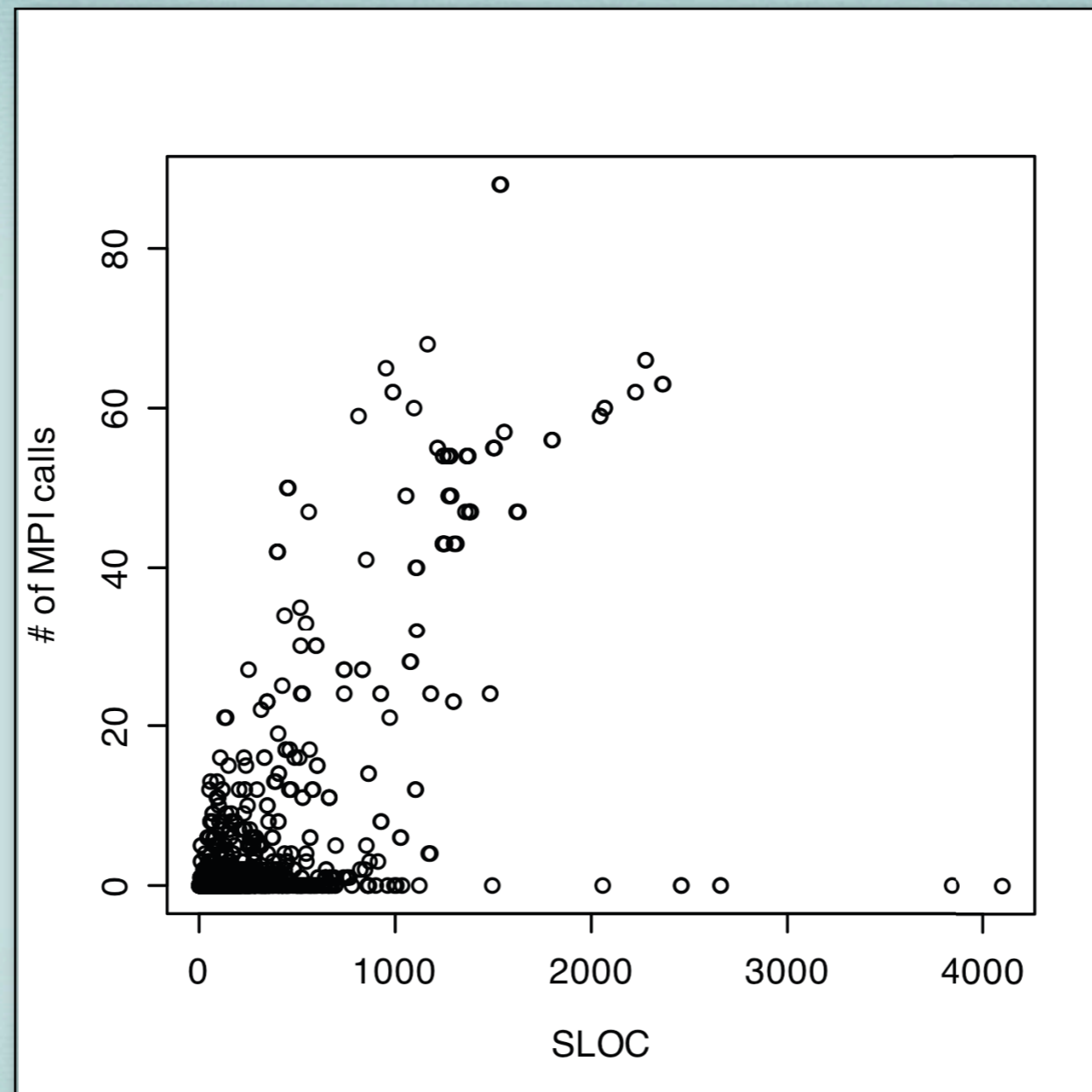
Productivity with MPI

TABLE II. AMOUNT OF MPI CODE

		# of files	SLOC
With PARAMESH	MPI	471 files	213,397 SLOC
	Total	2625 files	406,207 SLOC
	Percentage	<i>17.9%</i>	<i>52.5%</i>
Without PARAMESH	MPI	145 files	23,335 SLOC
	Total	1925 files	159,779 SLOC
	Percentage	<i>7.5%</i>	<i>14.6%</i>

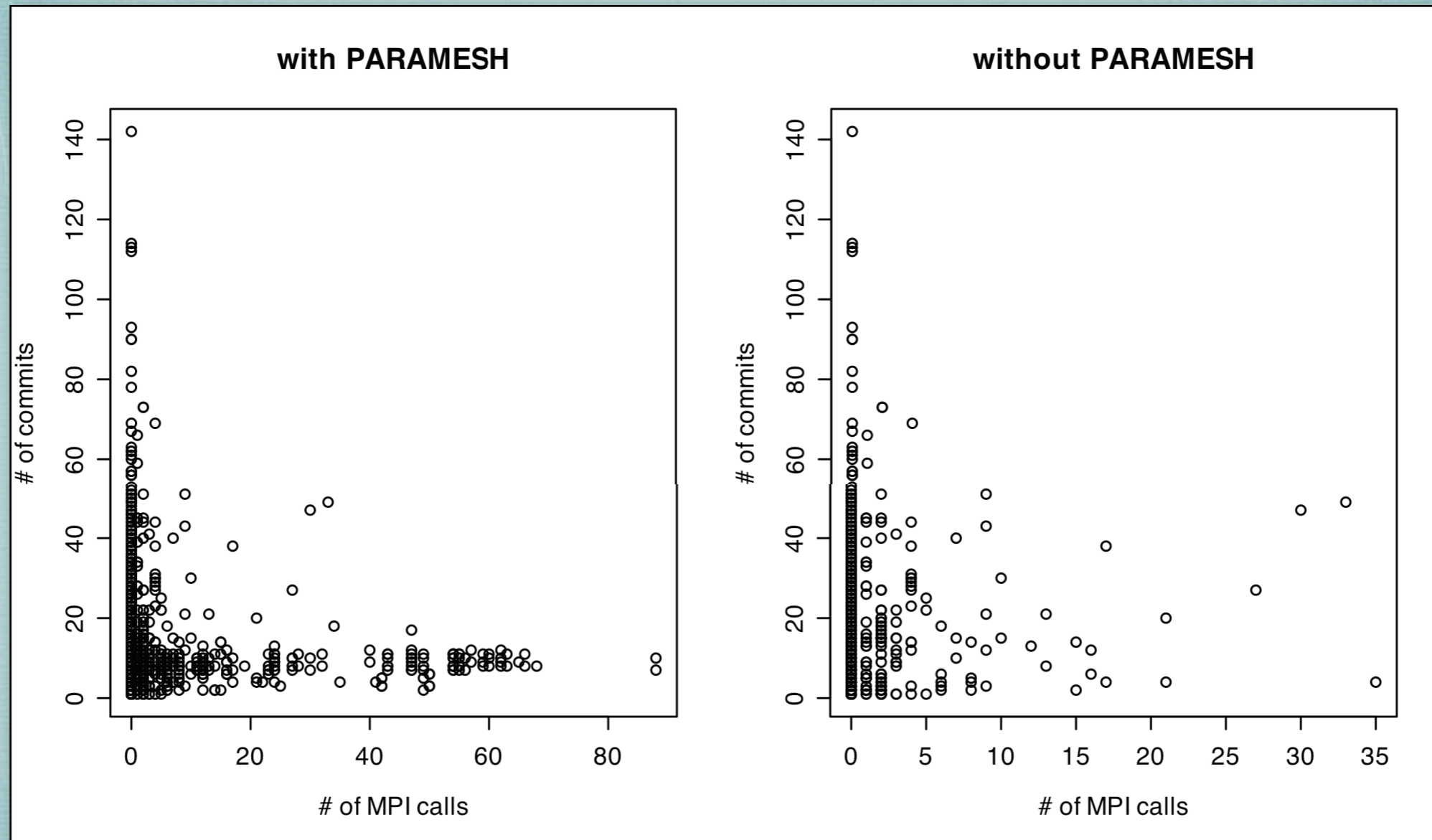
Lorin Hochstein, Forrest Shull, Lynn B. Reid. The Role of MPI in Development Time: A Case Study, In *Proceedings of the 2008 ACM/IEEE Conference on High Performance Networking and Computing (SC '08)*, 2008.

Productivity with MPI: MPI Calls vs SLOC



Lorin Hochstein, Forrest Shull, Lynn B. Reid. **The Role of MPI in Development Time: A Case Study**, In *Proceedings of the 2008 ACM/IEEE Conference on High Performance Networking and Computing (SC '08)*, 2008.

Productivity with MPI: MPI Calls vs Commits



Lorin Hochstein, Forrest Shull, Lynn B. Reid. **The Role of MPI in Development Time: A Case Study**, In *Proceedings of the 2008 ACM/IEEE Conference on High Performance Networking and Computing (SC '08)*, 2008.

Productivity with MPI: MPI Calls vs Commits

TABLE III. COMMITS RELATED TO BUG-FIXING

	# of commits
Non-MPI	2366 (69.9%)
MPI	1021 (30.1%)
Total	3387 (100%)

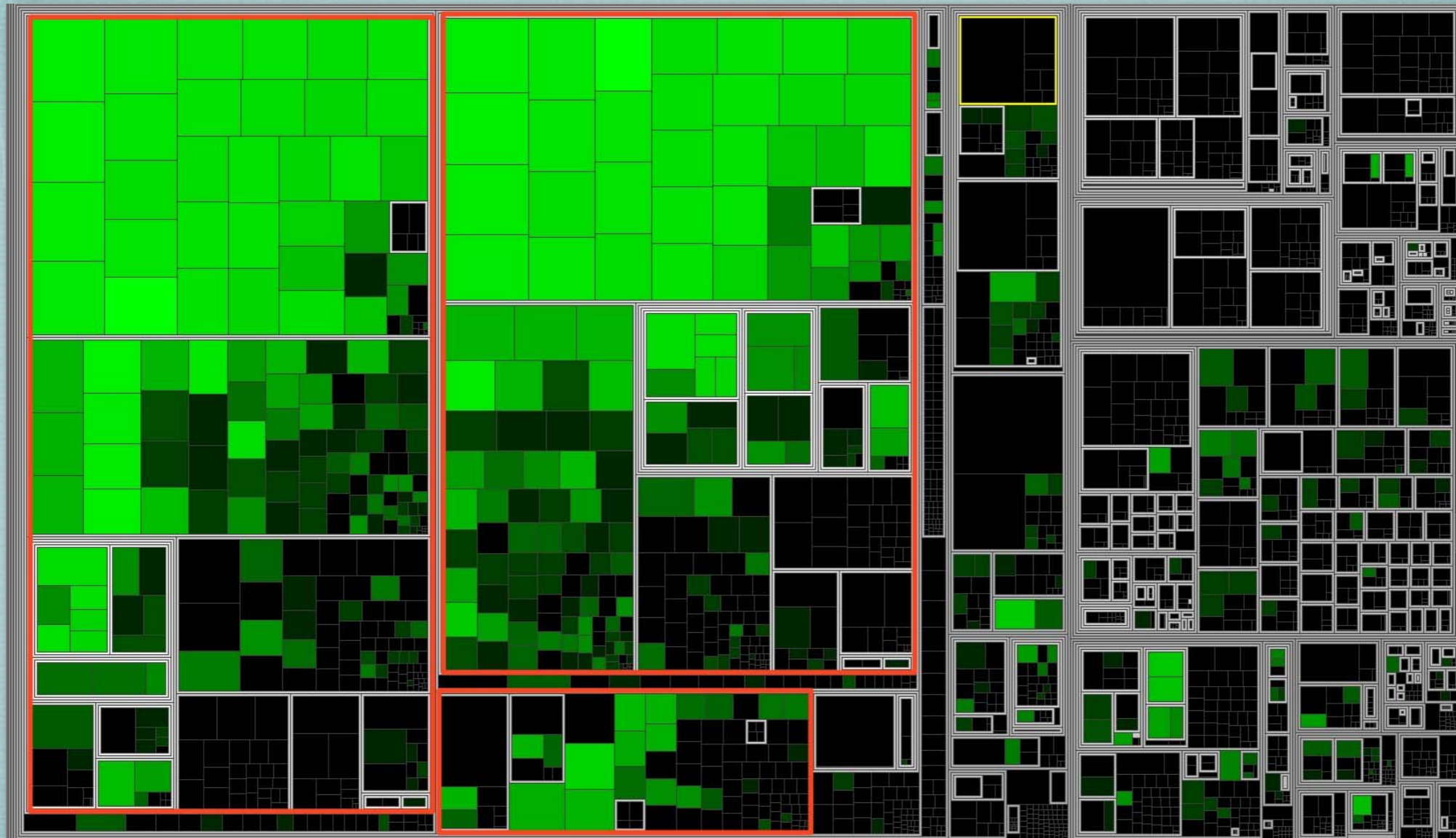
TABLE IV. COMMITS ACROSS DEVELOPERS

Developer	# of source commits	# of MPI commits	% MPI-related development
A	666 (16.2%)	112 (12.3%)	16.8%
B	630 (15.3%)	110 (12.1%)	17.5%
C	557 (13.6%)	162 (17.9%)	29.1%
D	484 (11.8%)	81 (8.9%)	16.7%
E	358 (8.7%)	36 (4.0%)	10.1%
F	286 (7.0%)	43 (4.7%)	15.0%
G	246 (6.0%)	41 (4.5%)	16.7%
H	241 (5.9%)	122 (13.5%)	50.6%
I	115 (2.8%)	81 (8.9%)	70.4%
J	102 (2.5%)	19 (2.1%)	18.6%
Total	4110	1220	

Lorin Hochstein, Forrest Shull, Lynn B. Reid. The Role of MPI in Development Time: A Case Study, In *Proceedings of the 2008 ACM/IEEE Conference on High Performance Networking and Computing (SC '08)*, 2008.

Productivity with MPI: Code Layout

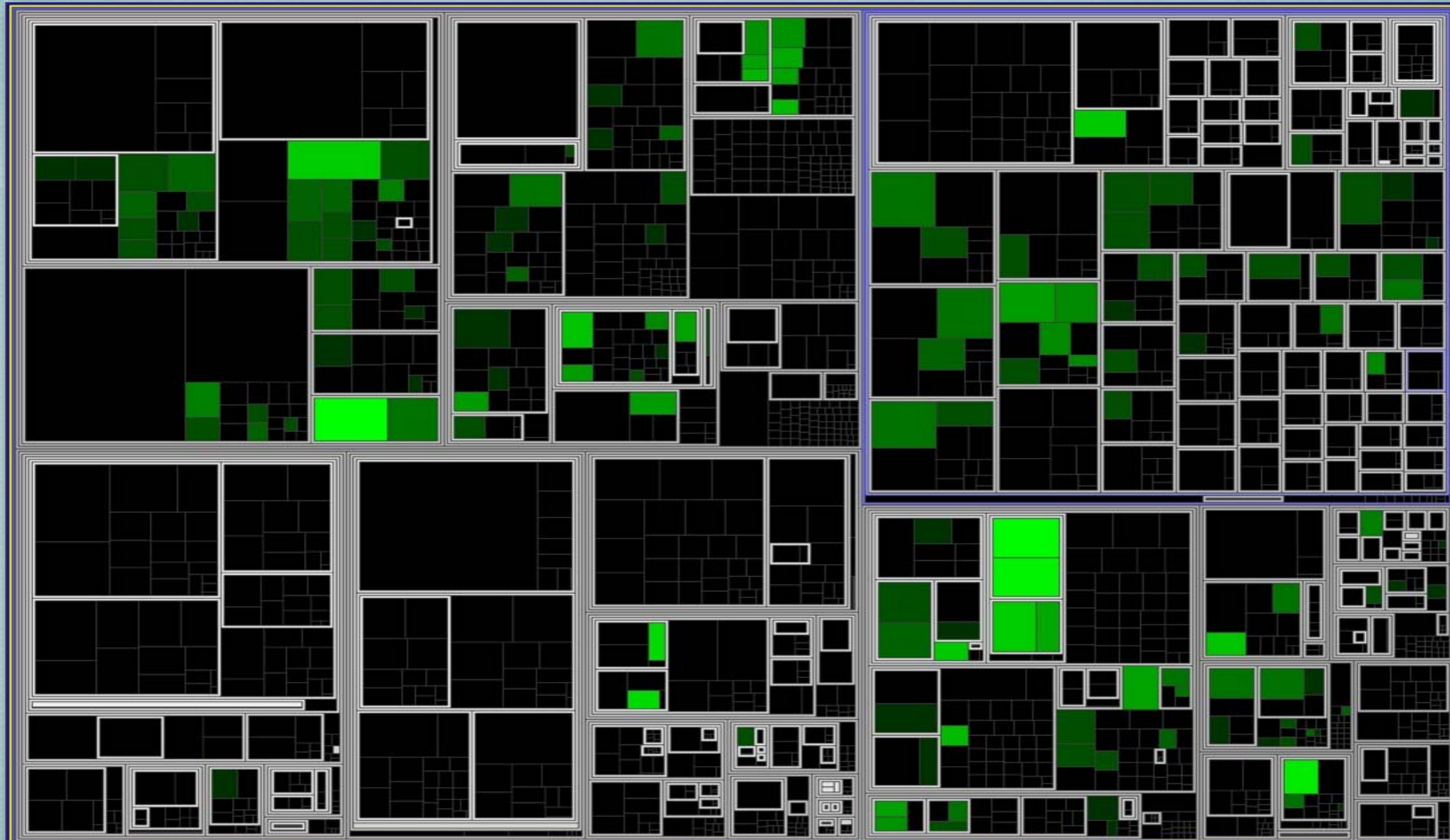
(with PARAMESH)



Lorin Hochstein, Forrest Shull, Lynn B. Reid. **The Role of MPI in Development Time: A Case Study**, In *Proceedings of the 2008 ACM/IEEE Conference on High Performance Networking and Computing (SC '08)*, 2008.

Productivity with MPI: Code Layout

(with PARAMESH)



Lorin Hochstein, Forrest Shull, Lynn B. Reid. **The Role of MPI in Development Time: A Case Study**, In *Proceedings of the 2008 ACM/IEEE Conference on High Performance Networking and Computing (SC '08)*, 2008.

Productivity with MPI: Conclusion

VII. CONCLUSION

In this paper, we have examined the role of MPI on a large-scale HPC code development project, and characterized the degree to which coding activities deal with MPI specifically. We addressed the issues inherent in any study of software engineering issues based on archaeological data by applying a rigorous case study methodology using triangulation: combining different measures to provide insight on the same topic. The general agreement among our measures (number of files, number of SLOC, number of commits, number of debugging activities) provides confidence that the results, showing that MPI-specific issues make up a small percentage of the overall coding activities, are indicative of real phenomena.

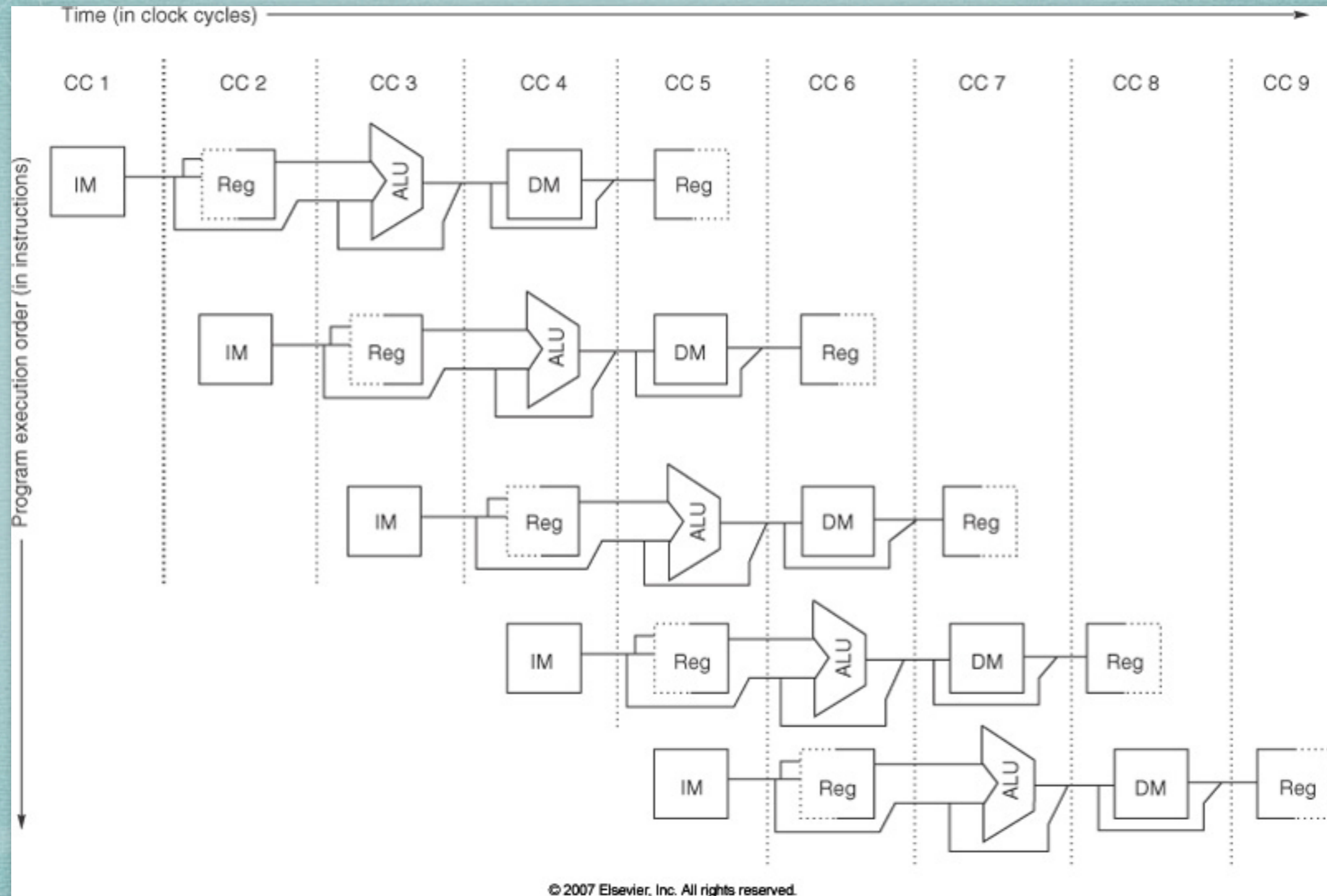
Lorin Hochstein, Forrest Shull, Lynn B. Reid. **The Role of MPI in Development Time: A Case Study**, In *Proceedings of the 2008 ACM/IEEE Conference on High Performance Networking and Computing (SC '08)*, 2008.

RECAP

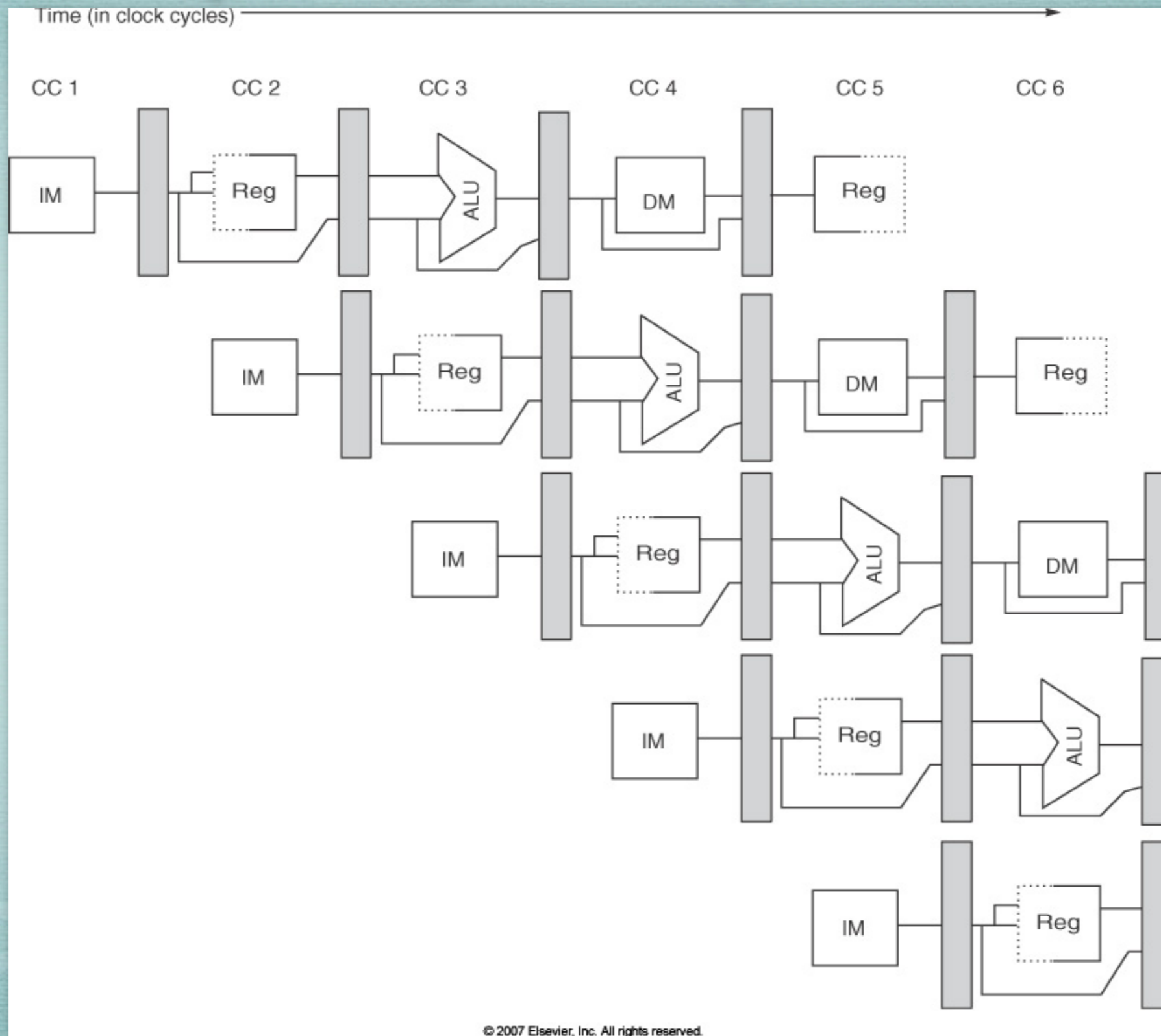
Recap

- ILP
- Exploiting ILP
- Dynamic scheduling
- Thread-level Parallelism
- Memory Hierarchy
- Other topics through student presentations
- Parallel programming and productivity

ILP: Pipelining

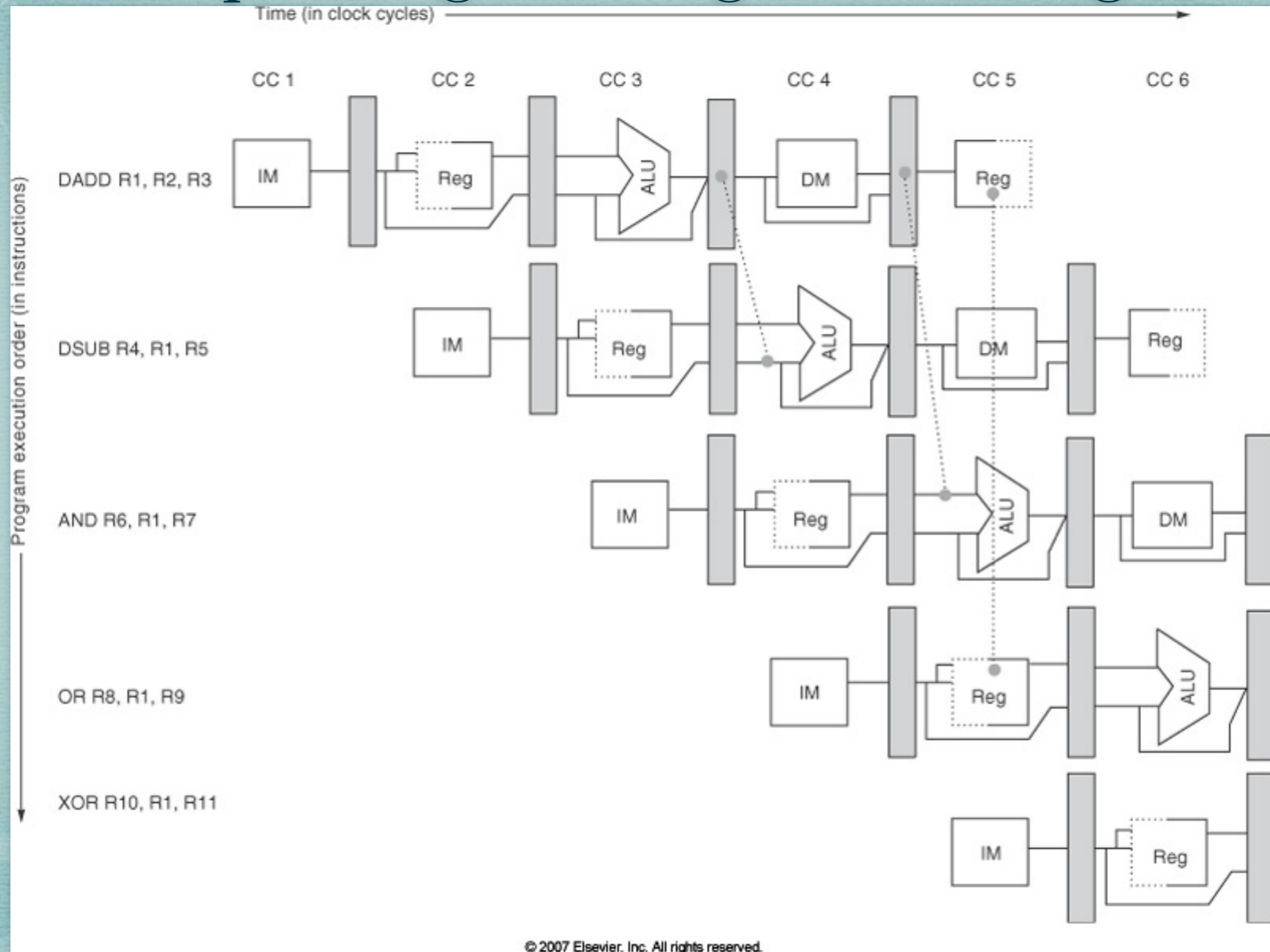


Pipelining: Adding Latches



© 2007 Elsevier, Inc. All rights reserved.

Pipelining: Adding Forwarding

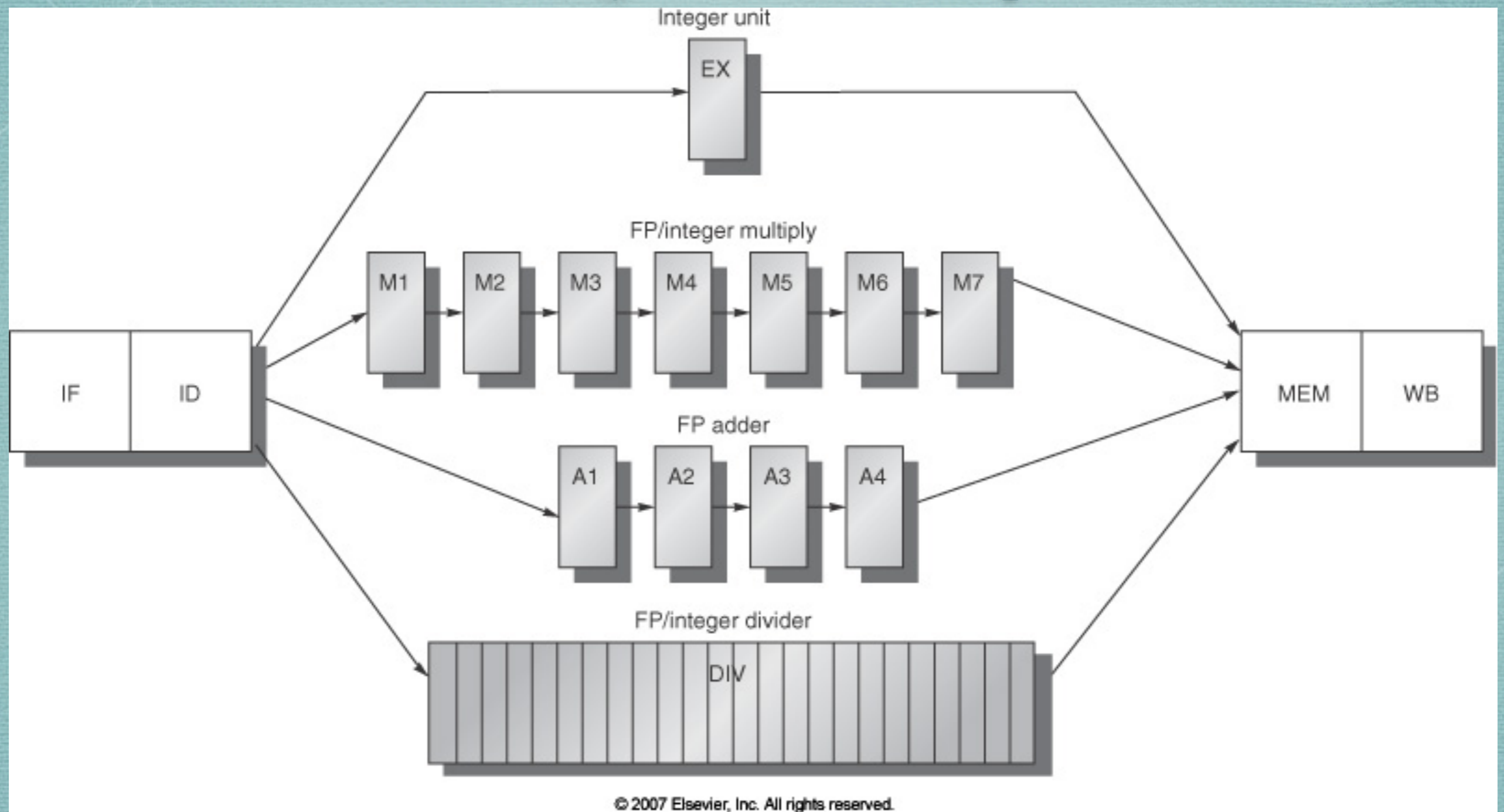


© 2007 Elsevier, Inc. All rights reserved.

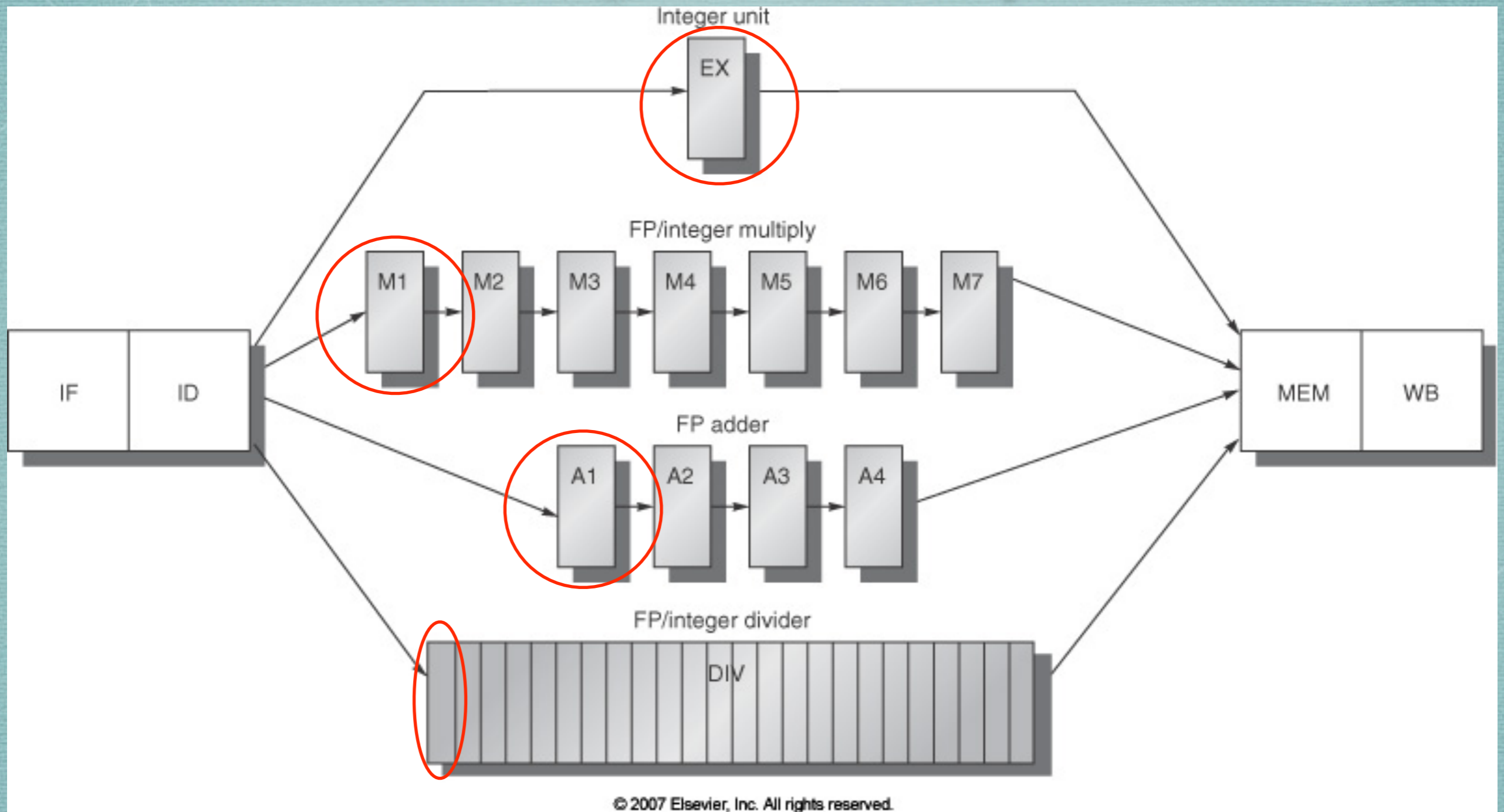
Pipelining: Adding Branch Delay Slots



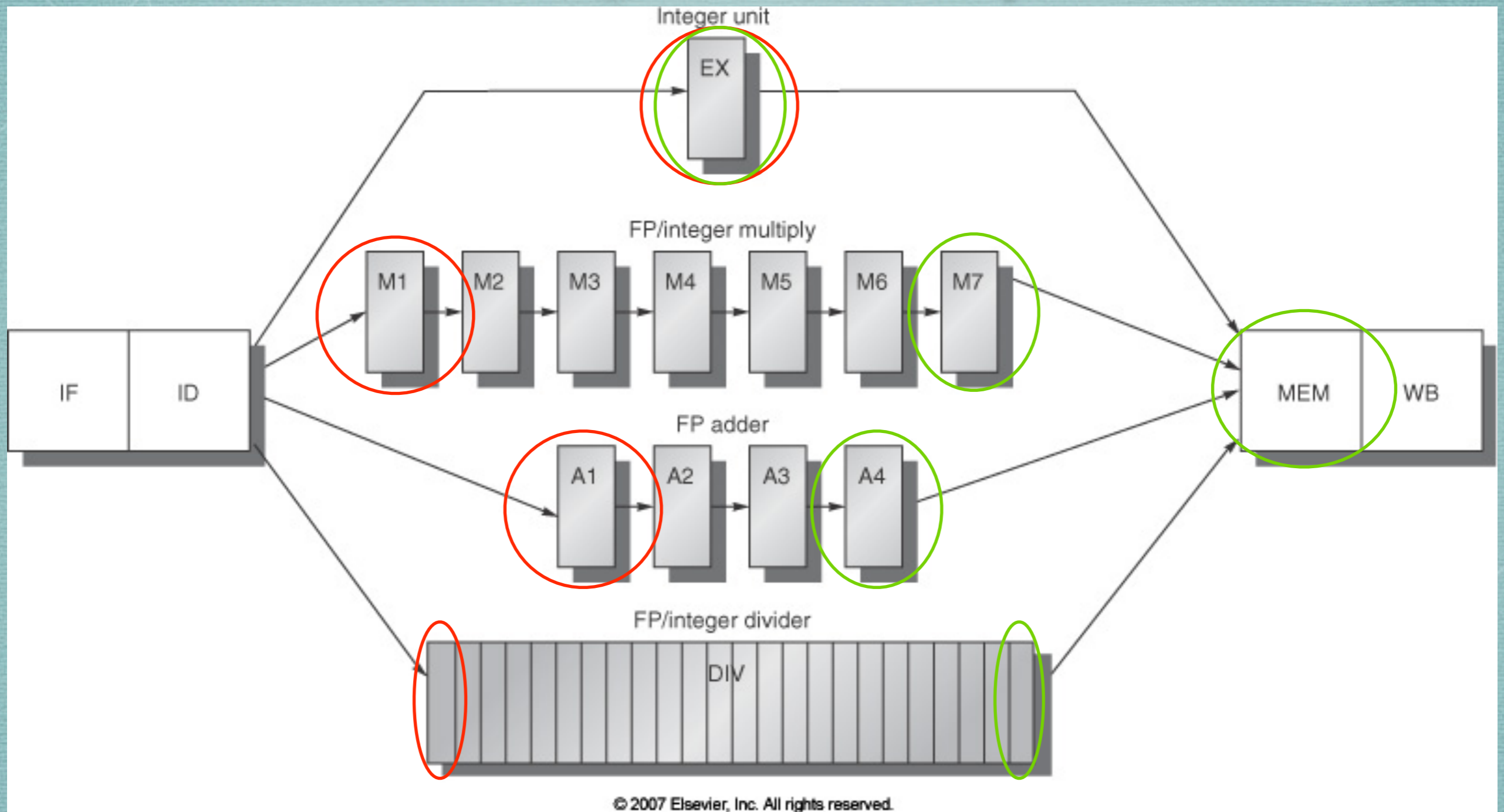
Extending the Basic Pipeline



Extending the Basic Pipeline



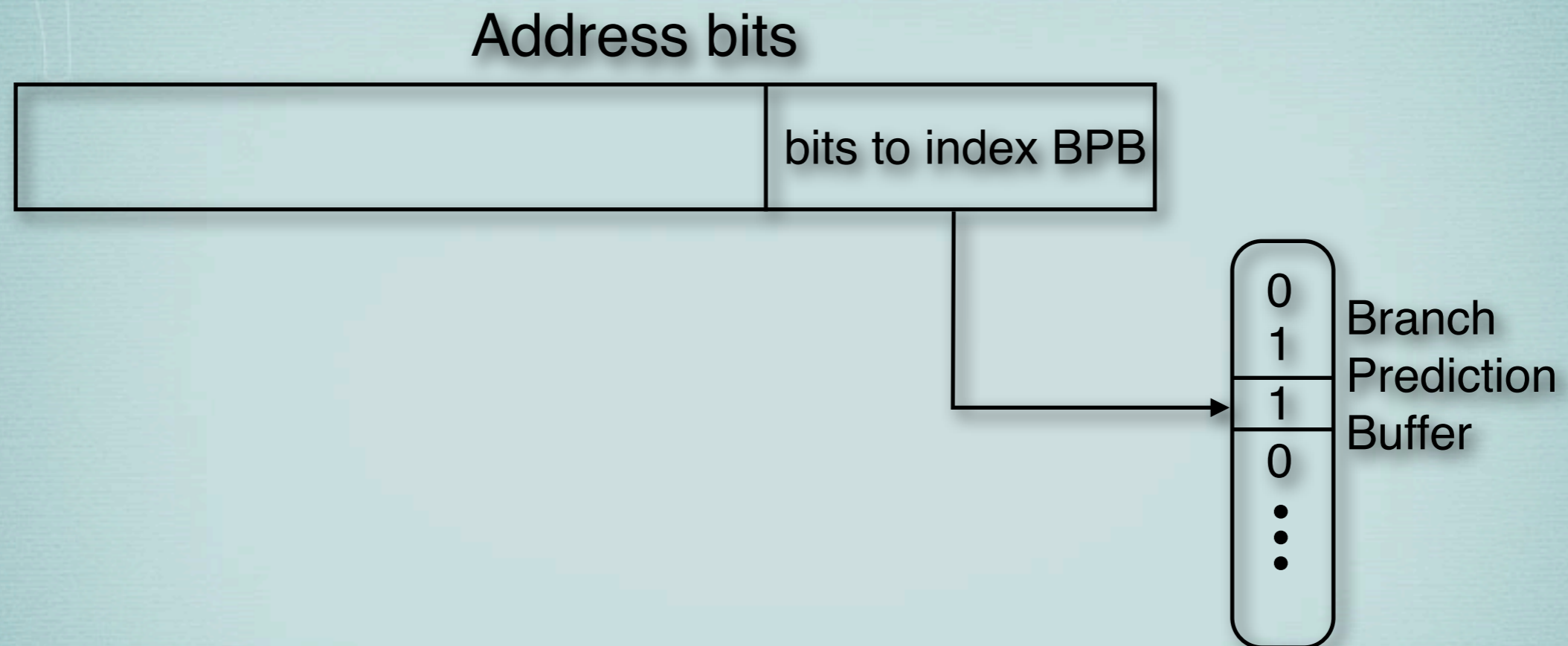
Extending the Basic Pipeline



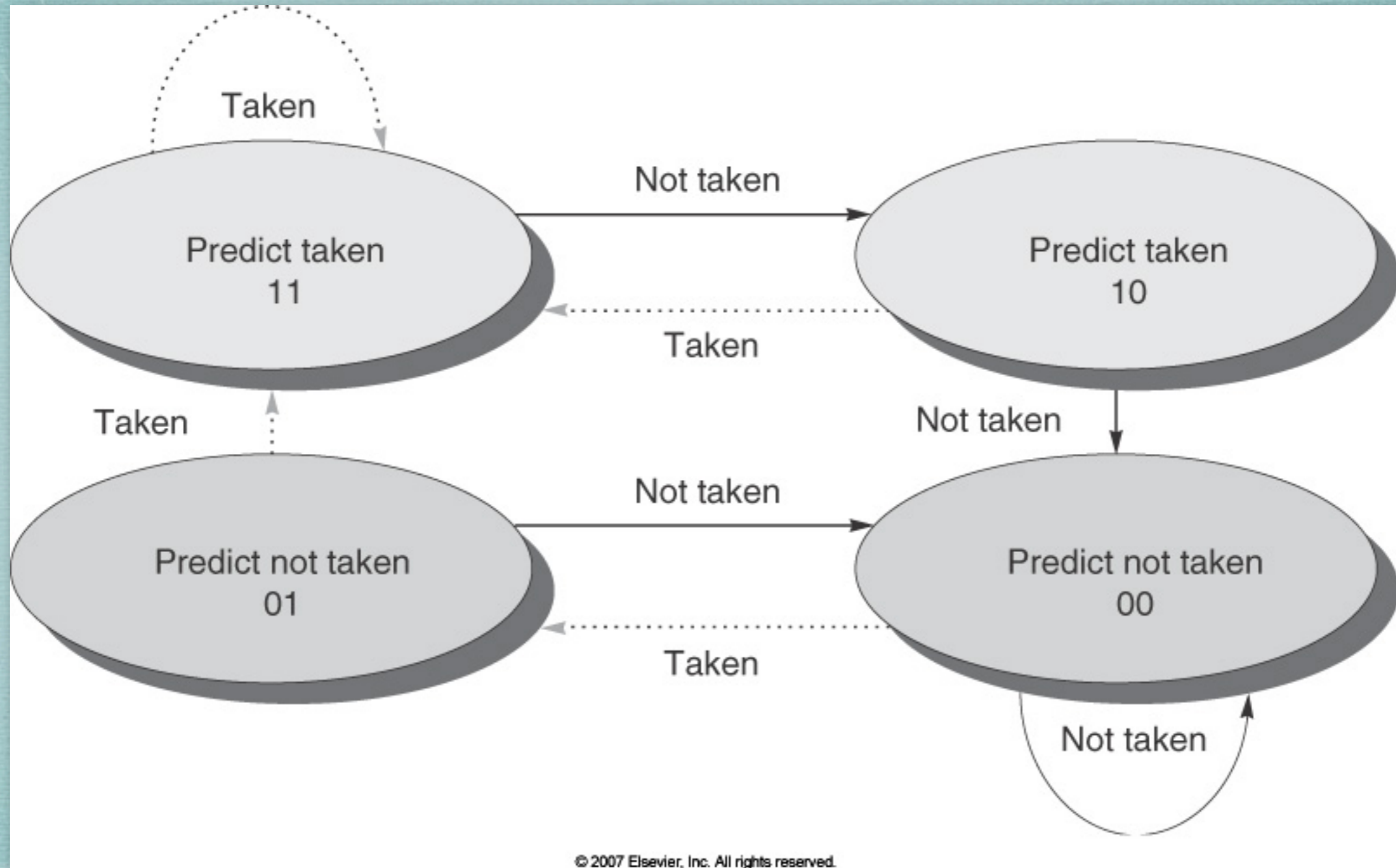
Exploiting ILP Through Compiler Techniques

- Loop unrolling
- Making use of branch delayed slots
- Static branch prediction
- Loop fusion
- Unroll and jam
- ...

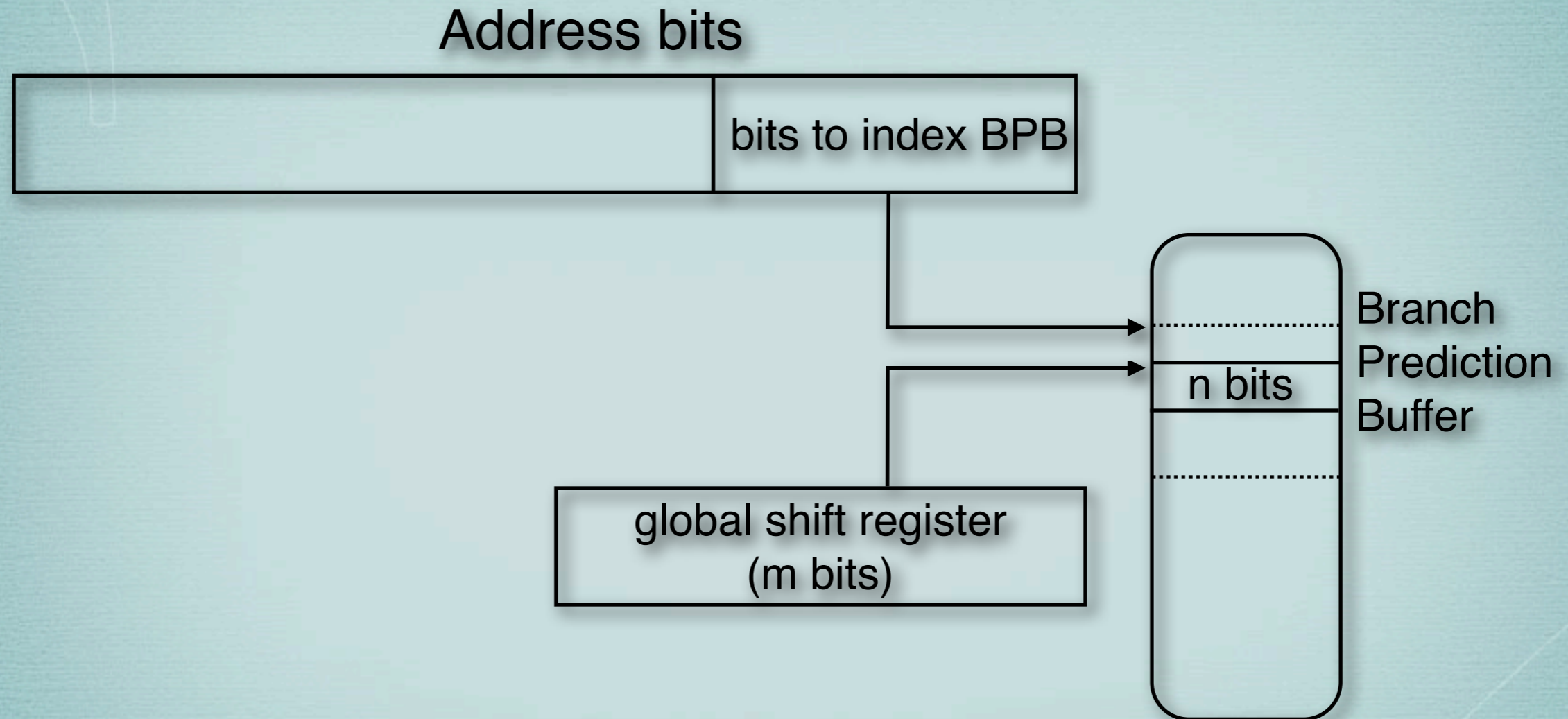
Dynamic Branch Prediction



Two-bit Branch Predictor

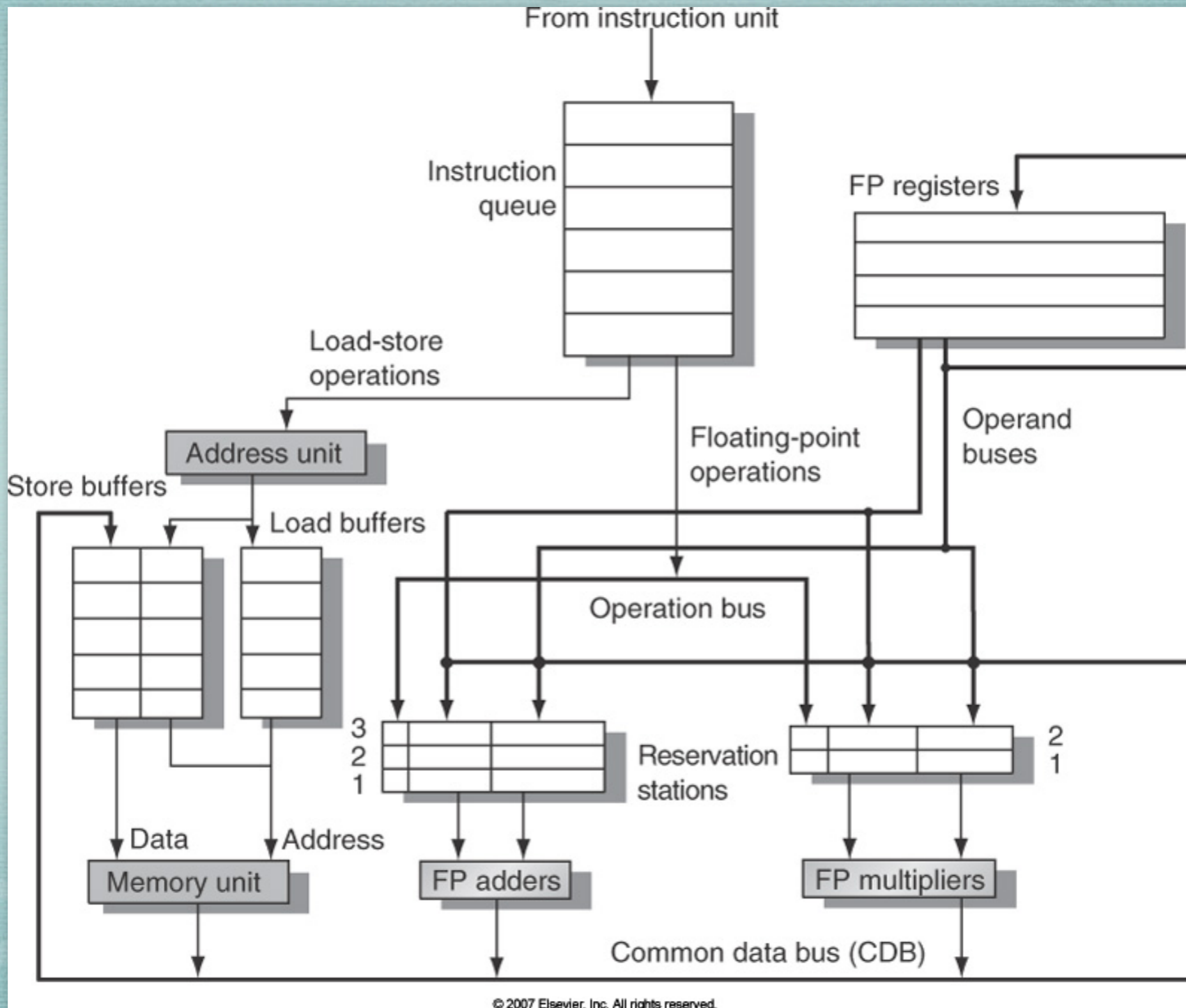


General n-bit Correlating Branch Predictors



Use Branch Target Buffers (BTBs) for caching branch targets

Dynamic Scheduling: Tomasulo's Approach



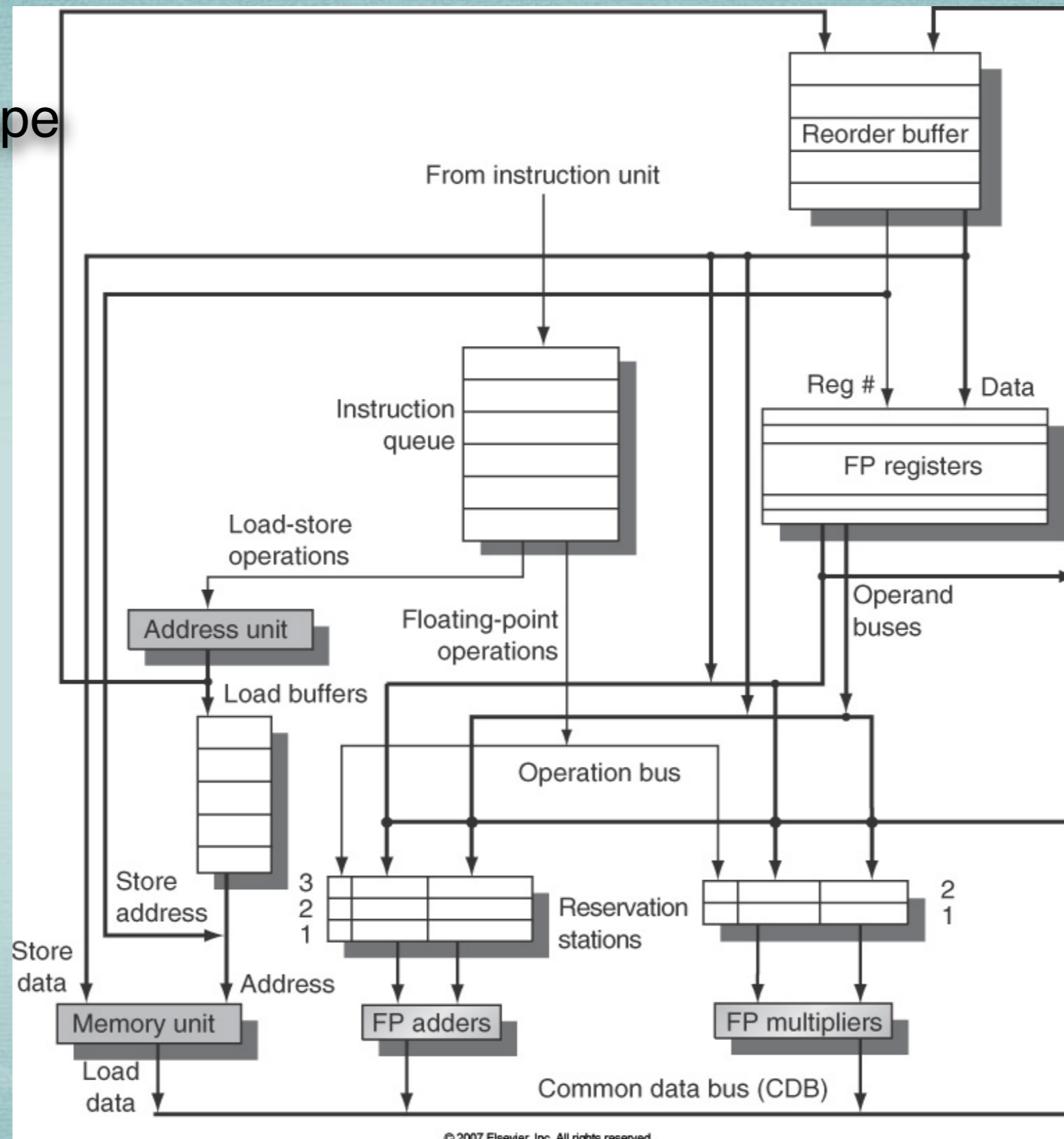
Tomasulo's Approach: Observations

- RAW hazards handled by waiting for operands
- WAR and WAW hazards handled by register renaming
 - ★ only WAR and WAW hazards between instructions currently in the pipeline are handled; is this a problem?
 - ★ larger number of hidden names reduces name dependences
- CDB implements forwarding

Tomasulo's Approach + Speculation

Fields in ROB

1. Instruction type
2. Destination
3. Value
4. Ready



Observations on Speculation

- Speculation enables precise exception handling
 - ★ defer exception handling until instruction ready to commit
- Branches are critical to performance
 - ★ prediction accuracy
 - ★ latency of misprediction detection
 - ★ misprediction recovery time
- Must avoid hazards through memory
 - ★ WAR and WAW already taken care of (how?)
 - ★ for RAW
 - * don't allow load to proceed if an active ROB entry has Destination field matching with A field of load
 - * maintain program order for effective address computation (why?)

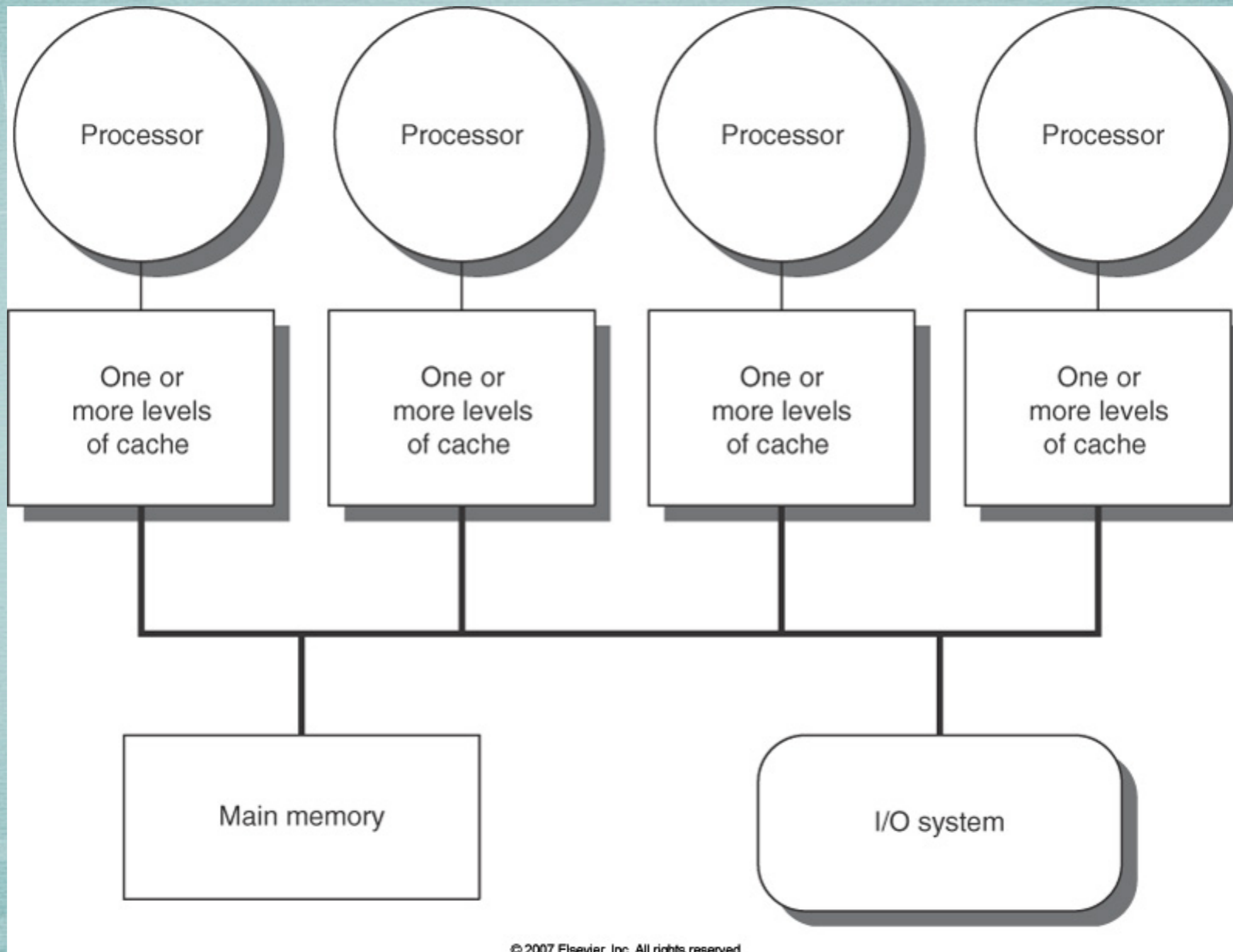
Multiple Issue Processor Types

Common name	Issue structure	Hazard detection	Scheduling	Distinguishing characteristic	Examples
Superscalar (static)	dynamic	hardware	static	in-order execution	mostly in the embedded space: MIPS and ARM
Superscalar (dynamic)	dynamic	hardware	dynamic	some out-of-order execution, but no speculation	none at the present
Superscalar (speculative)	dynamic	hardware	dynamic with speculation	out-of-order execution with speculation	Pentium 4, MIPS R12K, IBM Power5
VLIW/LIW	static	primarily software	static	all hazards determined and indicated by compiler (often implicitly)	most examples are in the embedded space, such as the TI C6x
EPIC	primarily static	primarily software	mostly static	all hazards determined and indicated explicitly by the compiler	Itanium

Dyn. Scheduling+Multiple Issue+Speculation

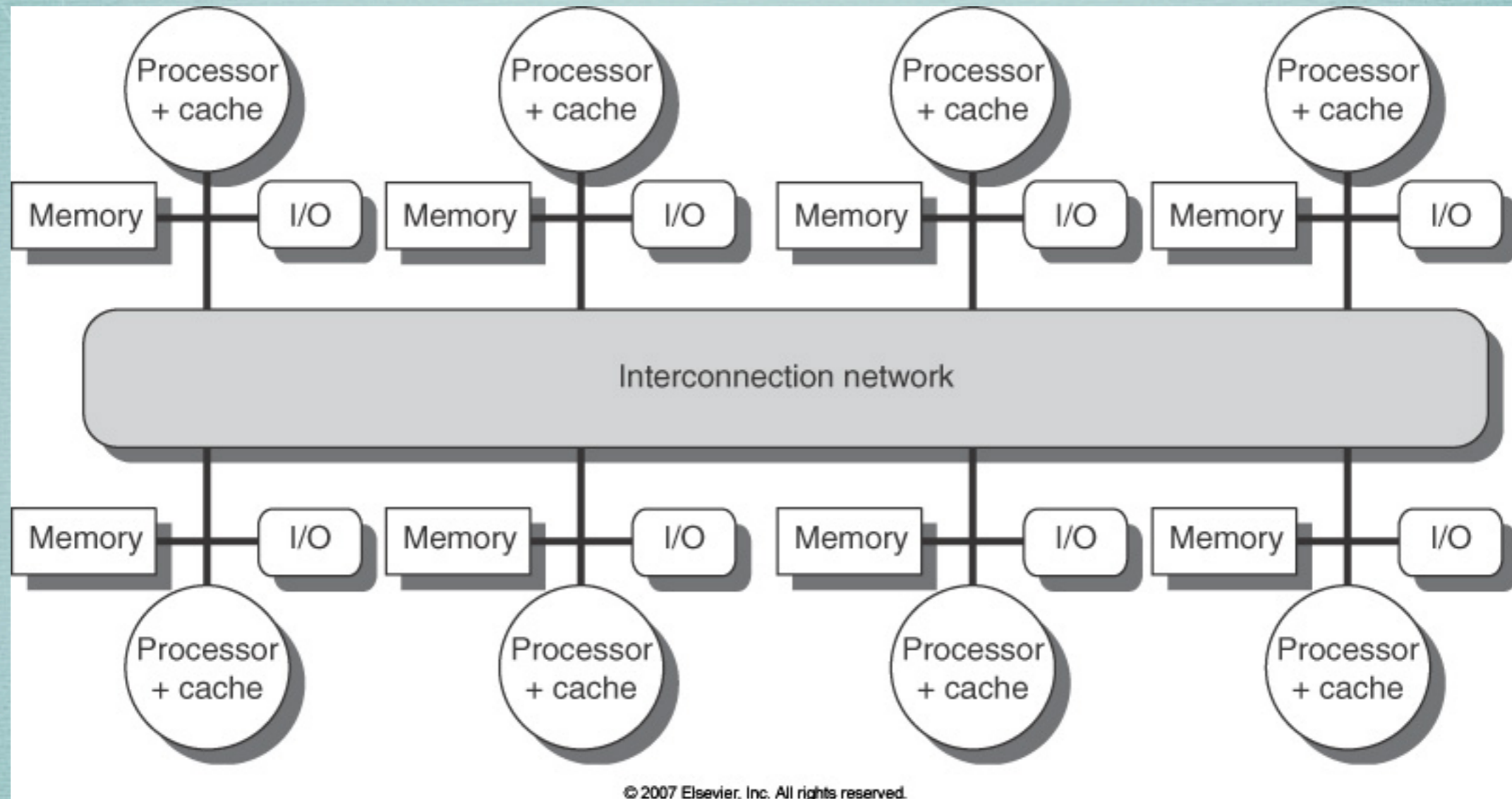
- Design parameters
 - ★ two-way issue (two instruction issues per cycle)
 - ★ pipelined and separate integer and FP functional units
 - ★ dynamic scheduling, but not out-of-order issue
 - ★ speculative execution
- Task per issue: assign reservation station and update pipeline control tables (i.e., control signals)
- Two possible techniques
 - ★ do the task in half a clock cycle
 - ★ build wider logic to issue any pair of instructions together
- Modern processors use both (4 or more way superscalar)

Shared-Memory Multiprocessors



© 2007 Elsevier, Inc. All rights reserved.

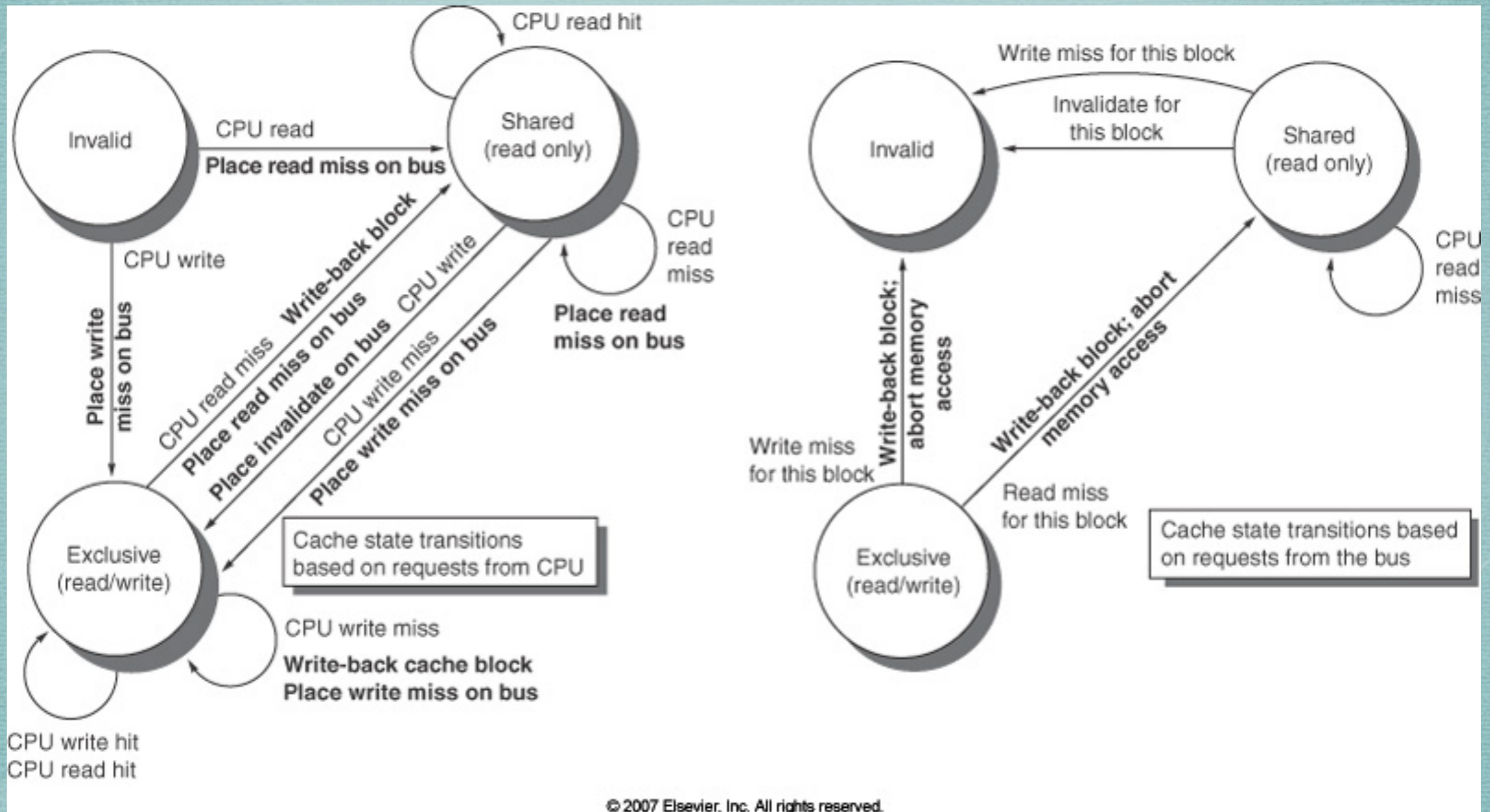
Distributed-Memory Multiprocessors



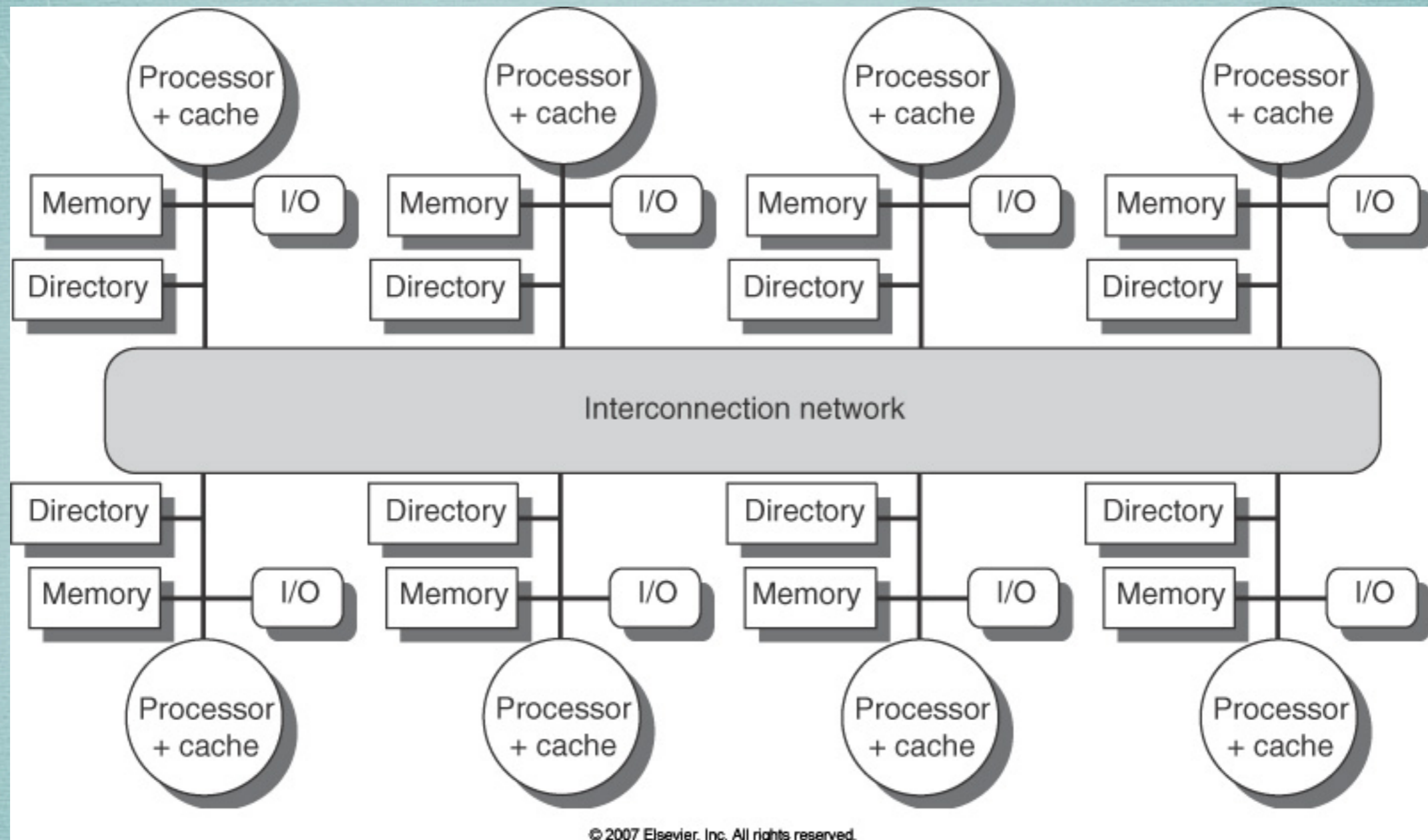
Other Ways to Categorize Parallel Programming



Write Invalidate Cache Coherence Protocol for Write-Back Caches



Distributed Memory+Directories



Other Topics

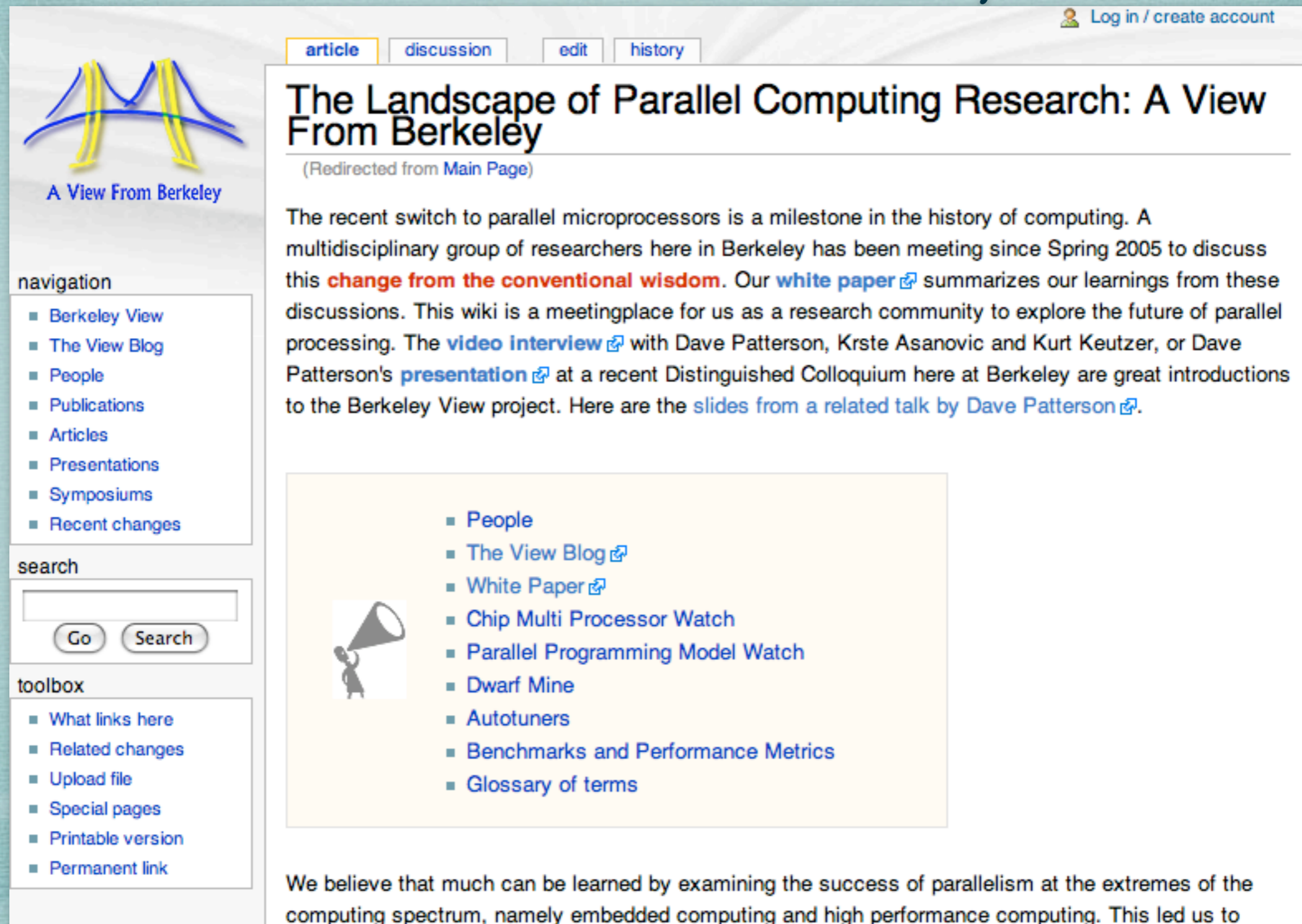
- x86 assembly programming
- VLIW / EPIC
- Vector processors
- Embedded systems
- Scientific applications
- GPUs and GPGPUs
- Interconnection networks
- Multi-stage interconnection networks
- Parallel graphs

WHAT'S NEXT?

Future

- Continued importance of parallel programming
 - ★ challenge: how to program multiprocessors
 - ★ role of programming languages and compilers
- Convergence or specialization?
 - ★ “standardization” of general purpose architecture
 - ★ migration of “special-purpose” CPUs for general use

Landscape of Parallel Computing Research: A View from Berkeley



The screenshot shows a Wikipedia article page. At the top right, there is a user login link: "Log in / create account". Below this are tabs for "article", "discussion", "edit", and "history". The article title is "The Landscape of Parallel Computing Research: A View From Berkeley", with a sub-note "(Redirected from Main Page)". The main text begins with: "The recent switch to parallel microprocessors is a milestone in the history of computing. A multidisciplinary group of researchers here in Berkeley has been meeting since Spring 2005 to discuss this **change from the conventional wisdom**. Our [white paper](#) summarizes our learnings from these discussions. This wiki is a meetingplace for us as a research community to explore the future of parallel processing. The [video interview](#) with Dave Patterson, Krste Asanovic and Kurt Keutzer, or Dave Patterson's [presentation](#) at a recent Distinguished Colloquium here at Berkeley are great introductions to the Berkeley View project. Here are the [slides from a related talk by Dave Patterson](#)." Below the text is a yellow box containing a list of links: "People", "The View Blog", "White Paper", "Chip Multi Processor Watch", "Parallel Programming Model Watch", "Dwarf Mine", "Autotuners", "Benchmarks and Performance Metrics", and "Glossary of terms". To the left of this list is a small icon of a person with a megaphone. On the far left, there is a sidebar with a logo "A View From Berkeley" (a stylized 'M' with blue and yellow lines), a "navigation" menu with links like "Berkeley View", "The View Blog", "People", "Publications", "Articles", "Presentations", "Symposiums", and "Recent changes"; a "search" box with "Go" and "Search" buttons; and a "toolbox" menu with links like "What links here", "Related changes", "Upload file", "Special pages", "Printable version", and "Permanent link".